Table of Contents

Running Jobs with PBS

Portable Batch System (PBS): Overview

All NAS facility supercomputers use the Portable Batch System (PBS) from Altair for batch job submission, job monitoring, and job management. Note that different systems may use different versions of PBS, so the available features may vary slightly from system to system.

Batch Jobs

Batch jobs run on compute nodes, not the front-end nodes. A PBS scheduler allocates blocks of compute nodes to jobs to provide exclusive access. You will submit batch jobs to run on one or more compute nodes using the *qsub* command from an interactive session on one of the front-end nodes (such as, pfe[1-12], bridge[1-2] for Pleiades or cfe2 for Columbia).

Normal batch jobs are typically run by submitting a script. A "jobid" is assigned after submission. When the resources you request become available, your job will execute on the compute nodes. When the job is complete, the PBS standard output and standard error of the job will be returned in files available to you.

Take carefully note when porting job submission scripts from systems outside of the NAS environment or between the Pleiades and Columbia supercomputers you may need to make changes to your existing scripts to make them work properly on these systems.

Interactive Batch Mode

PBS also supports an interactive batch mode, using the *qsub -I*, where the input and output is connected to the user's terminal, but the scheduling of the job is still under control of the batch system.

Queues

The available queues on different systems vary, but all typically have constraints on maximum wall-time and/or the number of CPUs allowed for a job. Some queues may also have other constraints or be restricted to serving certain users or groups. In addition, to ensure that each NASA mission directorate is granted their allocated share of resources at any given time, mission directorate limits (called "shares") are also set on Pleiades and Columbia.

See **man pbs** for more information.

Job Accounting

DRAFT

This article is being reviewed for completeness and technical accuracy.

Usage on the HECC machines at NAS, except for the front-end machines, is charged.

Starting May 1, 2011, the accounting unit is the Standard Billing Unit (SBU). The SBUs charged to a PBS job running on the compute node(s) is:

*SBU charged = Wall_Clock_Hours_Used * Number of MAUs * SBU Rate*

where the MAU represents the minimum allocatable unit of resources available through PBS. On Pleiades, an MAU is a node (with 8 cores for Harpertown and Nehalem-EP or 12 cores for Westmere in each node). On Columiba, an MAU has 4 cores. Charging is based on the number of MAUs allocated to a job, not how many cores are actually used during run-time. Once a user is allocated the resources, that user has exclusive access to those resources until the user's job completes or exceeds its requested wall-clock time.

The SBU rate for each of the NAS processors is outlined below:

Host	SBU Rate (per MAU)
Pleiades Westmere-EP nodes	1.00
Pleiades Nehalem-EP nodes	0.80
Pleiades Harpertown nodes	0.45
Columbia Itanium-2	0.18

In addition, charges on Columbia apply both to jobs that run successfully and those that are interrupted. Interrupted jobs are charged by taking the elapsed job time in hours, subtracting 1 hour, multiplying that by the number of MAUs used, and then deducting the resulting amount from the allocation. (Users are encouraged to have their applications checkpoint roughly every hour.)

In the near future, interruptions on Pleiades will be handled in a similar manner.

For example:

If you have a 24-hour job on Columbia that requires 16 MAUs (i.e., 64 cores), It has run for 12 hours and the system crashes. The accounting system will take the 12 hours, subtract 1 hour, and compute the SBUs (11 hours X 16 MAUs x 0.18 = 31.68 SBUs), which will then be subtracted from your allocation for your GID.

Job Accounting Utilities

DRAFT

This article is being reviewed for completeness and technical accuracy.

The job accounting utilities "acct_ytd" and "acct_query" can be used to obtain resource usage and charging information about your account, the accounts of other users on your project, and the project as a whole. Daily usage totals for each account are available for the current operational period.

- **acct_ytd**

  The "acct_ytd" command provides a year-to-date summary of accounting information for groups to which a user belongs. It will normally be accurate as of midnight the previous night, when accounting was last run.

  A number of parameters can be used with "acct_ytd", but the simpliest way is to type "acct_ytd" on a host without any parameters. This produces a line of output for each project you have access to on that host.

  ```
  %acct_ytd
  ```
  You can also specify the host group and/or a specific GID (for example, a0800).

  ```
  %acct_ytd -cpleiades a0800   %acct_ytd -ccolumbia a0800
  ```

  To find the allocations and usages of all your GIDs on all hosts, use the -call flag.

  ```
  %acct_ytd -call
  ```
  See **man acct_ytd** on Pleiades and Columbia for more information.
- **acct_query**

  The "acct_query" command searches and displays process-level billing records. This means that while totals over a period or for each day in a period are possible, you can also obtain detailed billing records for each process run in a period.

  For example, to see all the SBU usage, beginning June 1, 2010, ending July 1, 2010, for all projects and on all hosts by user zsmith:

  ```
  %acct_query -b06/01/10 -e07/01/10 -pall -call -uzsmith
  ```
  To see the current SBU usage for the operational year 2010 (defined as May 1, 2010 to May 1, 2011 for most mission directorates) for all projects and on all hosts by user zsmith:

  ```
  %acct_query -y10 -pall -call -uzsmith
  ```

Eligible hostnames include:

1. columbia21
2. columbia22
3. columbia23
4. columbia24
5. pbs1
6. pleiades (for Harpertown nodes)
7. pleiades_N (for Nehalem nodes)
8. pleiades_W (for Westmere nodes)

See **man acct_query** on Pleiades and Columbia for more information.

Multiple GIDs and Charging to a specific GID

DRAFT

This article is being reviewed for completeness and technical accuracy.

Each approved project is assigned a project id (GID). Members of a GID are authorized to use the resources allocated to that GID. For those users who have access to multiple GIDs, be aware that only one of those GIDs is considered your default.

Use the "groups" command to find which GIDs you are a member of. The following example shows that user zsmith is a member of the groups a0800, a0907, all, and e0720.

```
%groups zsmith
zsmith : a0800 a0907 all e0720
```

The first GID from the "groups" list should be your default GID. This can be verified through the /etc/passwd file. For example, the /etc/passwd file has an entry for user zsmith with the GID 20800 (which is the same as a0800, his default GID).

```
%grep zsmith /etc/passwd
zsmith:x:6666:20800:Z. Smith,,650-604-4444,:/u/zsmith:/bin/csh
```

When you use resources on the compute nodes through PBS jobs, SBUs are deducted from your default GID unless you specify otherwise. To charge resource usage to an alternative GID for a batch job, you can use the PBS flag "-W group_list=*account*" either in your script or on the "qsub" command line. For example:

```
#PBS -W group_list=a0907
```

or

```
%qsub -W group_list=a0907
```

Commonly Used PBS Commands

DRAFT

This article is being reviewed for completeness and technical accuracy.

man pbs provides a list of all PBS commands. The four most commonly used PBS commands, qsub, qstat, qdel and qhold, are briefly described below.

- **qsub**
    - Submit a batch job to the specified queue using a script

      ```
      %qsub -q queue_name job_script
      ```
      Common possibilities for *queue_name* at NAS include *normal, debug, long,* and *low*. When *queue_name* is omitted, the job is routed to the default queue, which is the *normal* queue.
    - Submit an interactive PBS job

      ```
      %qsub -I -q queue_name -lresource_list

      No job_script should be included when submitting an
      interactive PBS job.
      ```
      The *resource_list* typically specifies the number of nodes, cpus, amount of memory and walltime needed for this job. The following example shows a request for Pleides with 2 nodes, 8 cpus per node, and a walltime limit of 3 hours.

      ```
      %qsub -I -lselect=2:ncpus=8,walltime=3:00:00
      ```
      See **man pbs_resources** for more information on what resources can be specified. If -l*resource_list* is omitted, the default resources for the specified queue is used. When *queue_name* is omitted, the job is routed to the default queue, which is the *normal* queue.
- **qstat**
    - Display queue information

      ```
      %qstat -Q queue_name
      %qstat -q queue_name
      $qstat -fQ queue_name
      ```
      These commands display in different formats all the queue available on the systems, their constraints and status. The *queue_name* is optional.
    - Display job status

◊ Display all jobs in any status (running, queued, held)

```
%qstat -a
```

◊ Display all running or suspended jobs

```
%qstat -r
```

◊ Display the execution hosts of the running jobs

```
%qstat -n
```

◊ Display all queued, held or waiting jobs

```
%qstat -i
```

◊ Display jobs that belong to the specified user

```
%qstat -u user_name
```

◊ Display any comment added by the administrator or scheduler

```
%qstat -s
```

This option is typically used to find clues of why a job has not started running.

◊ Display detailed information about a specific job

```
%qstat -f job_id
```

◊ Display status informaton for finished jobs (within the past 7 days)

```
%qstat -xf job_id
%qstat -xu user_id
```

This option is only available in newer version of PBS, which has been installed on Pleiades, but not on Columbia.

```
Some of these flags can be combined when checking the job status.
```

- **qdel**

  Delete a job

  ```
  %qdel job_id
  ```

- **qhold**

  Hold a job

  ```
  %qhold job_id
  ```

  Only the job owner or a system administrator with su or root privilege can place a hold on a job. The hold can be released using the "qrls" command.

For more detailed information on each command, see their corresponding **man pages**.

The *devel* queue on Pleiades is served by a non-default PBS server, pbspl3, and the syntax for qsub, qstat, and qdel jobs in the *devel* queue needs to include pbspl3. Read this article for more information.

Commonly Used QSUB Options in PBS Scripts or in the QSUB Command Line

DRAFT

This article is being reviewed for completeness and technical accuracy.

The "qsub" options can be read from the PBS directives of a PBS *job_script* or from the qsub command line. For a complete list of available options, see **man qsub**. The more commonly used ones are listed below.

-S *shell_name*
: Specifies the shell that interprets the job script

-V
: Declares that all environment variables in the qsub command's environment are to be exported to the batch job

-v *variable_list*
: Lists environment variables to be exported to the job

-q *queue_name*
: Defines the destination of the job. The common possibilities for *queue_name* on Pleides and Columbia include *normal, debug, long,* and *low*

  The *devel* queue on Pleiades is served by a non-default PBS server, pbspl3, and the syntax for qsub jobs to the *devel* queue needs to include pbspl3. Read this article for more information.

-l *resouce_list*
: Specifies the resources that are required by the job and establishes a limit to the amount of resources that can be consumed. Commonly used resource items are slect, ncpus, walltime, and memory. See **man pbs_resources** for a complete list of available resources.

-e *path*
: Directs the standard error output produced by the request to the stated file path

-o *path*
: Directs the standard output produced by the request to the stated file path.

-j *join*
: Declares that the standard output and error streams of the job should be merged (joined). The values for *join* can be:

oe standard output and error streams are merged in the standard output file
eo standard error and output streams are merged in the standard error file

-m *mail_options*

Defines the set of conditions under which the execution server will send mail message about the job. See **man qsub** for a list of mail_options.

-N *name*

Declares a name for the job

-W *addl_attributes*

Allows for the specification of additional job attributes
The most common ones are
-W group_list=*g_list* specifies the group the job runs under
-W depend=afterany:*job_ID.server_name.nas.nasa.gov* (for example, 12345.pbspl1.nas.nasa.gov) submits a job which is to be executed after *job_ID* has finished with any exit status
-W depend=afterok:*job_ID.server_name.nas.nasa.gov* (for example, 12345.pbspl1.nas.nasa.gov) submits a job which is to be executed after *job_ID* has finished with no errors

-r *y|n*

Declares whether the job is rerunnable

The top of a PBS *job_script* contains PBS directives, each of which begins with the string "#PBS". Here is an example for use on Pleiades.

```
#PBS -S /bin/csh
#PBS -V
#PBS -q long
#PBS -lselect=2:ncpus=8:mpiprocs=4:model=har,walltime=24:00:00
#PBS -j oe
#PBS -o /nobackup/zsmith/my_pbs_output
#PBS -N my_job_name
#PBS -m e
#PBS -W group_list=a0907
#PBS -r n
```

The resources and/or attributes set using options to the "qsub" command line override those set in the directives in the PBS *job_script*.

New Features in PBS

DRAFT

This article is being reviewed for completeness and technical accuracy.

Some of the new features relevant to users are listed below:

- Estimate job start times (version 10.4)

  PBS can estimate the start time for jobs. To show the estimated start times (in the *Est Start* field), use

  ```
  %qstat -T
  ```

  ```
  This feature is still under testing by NAS system
  administrators and is not yet available to users.
  ```

- Show the processor model (version 10.4)

  Processor model (for example, Harpertwon, Nehalem-EP, and Westmere-EP) can be displayed with

  ```
  %qstat -W o=+model
  ```

- Show job history (version 10.1)

  Use the PBS "-x" option to obtain job history information, including the submission parameters, start/end time, resources used, etc., for jobs that finished executation, were deleted or are still running.

  The job history for finished jobs is preserved for a specific duration. After the duration has expired, PBS deletes the job history information and it is no longer available. Currently, the duration is set to be **7 days** on Pleiades.

  ```
  %qstat -fx job_id
  ```

- Advance and Standing reservations (version 9.2)

  An advance reservation can be made for a set of resources for a specified time. The reservation is only available to a specific user or group of users.

  A standing reservation is an advance reservation which recurs at specified times. For example, the user can reserve 8 nodes every Wednesday from 5pm to 8pm, for the next month.

The reservation is made using the "pbs_rsub" command. PBS either confirms that the reservation can be made, or rejects the request. Once he reservation is confirmed, PBS creates a queue for the reservation's jobs. Jobs are then submitted to this queue.

The following example shows the creation of an advance reservation asking for 1 node with 8 cpus, a start time of 11:30 and a duration of 30 minutes.

```
%pbs_rsub -R 1130 -D 00:30:00 -l select=1:ncpus=8
```

A reservation can be deleted using the "pbs_rdel" command.

For more information, see **man pbs_rsub** and **man pbs_rdel**.

```
Requests to use advance and standing reservations must be
approved by NAS management. Only staff with special privilege
can create the reservations for users.
```

Checkpointing and Restart

DRAFT

This article is being reviewed for completeness and technical accuracy.

None of the NAS HEC systems has an automatic checkpoint capability made available by the operating system. For jobs that need lots of resources and/or long wall-time, you should have a checkpoint/restart capability implemented in the source code or job script.

PBS automatically restarts unfinished jobs after system crashes. If you do not want PBS to restart your job, make sure to add the following in your PBS script:

```
#PBS -r n
```

PBS Environment Variables

DRAFT

This article is being reviewed for completeness and technical accuracy.

There are a number of environment variables provided to the PBS job. Some are taken from the user's environment and carried with the job. Others are created by PBS. Still others can be explicitly created by the user for exclusive use by PBS jobs. All PBS-provided environment variable names start with the characters "PBS_". Some are then followed by a capital O ("PBS_O_") indicating that the variable is from the job's originating environment (i.e. the user's).

The following lists a few useful PBS environment variables.

PBS_O_WORKDIR
: contains the name of the directory from which the user submitted the PBS job

PBS_O_PATH
: value of **PATH** from submission environment

PBS_JOBID
: contains the PBS job identifier

PBS_JOBDIR
: pathname of job-specific staging and execution directory

PBS_NODEFILE
: contains a list of vnodes assigned to the job

TMPDIR
: The job-specific temporary directory for this job
: defaults to /tmp/pbs.*job_id* on the vnodes

PBS Scheduling Policy

DRAFT

This article is being reviewed for completeness and technical accuracy.

This article gives a simplified explanation of the PBS scheduling policy on Pleiades and Columbia

PBS scheduling policies change frequently, in response to varying demands and workloads. The current policy (March 1, 2011), simplified, states that jobs are sorted in the following order: current mission directorate CPU use, job priority, queue priority, and job size (wide jobs first).

In each scheduling cycle, PBS examines the jobs in sorted order, starting a job if it can. If the job cannot be started immediately, it is either scheduled around or simply bypassed for this cycle.

There are numerous reasons why jobs won't start, such as:

- The queue is at its running job limit
- You are at your running job limit
- The queue is at its CPU limit
- The mission directorate is at its CPU share limit and the job cannot borrow from another mission
- Not enough CPUs are available

Notice that a high-priority job might be blocked by some limit, while a lower priority job, from a different user or asking for fewer resources, might not be blocked.

If your job is waiting in the queue, use the following commands to get some information about why it has not started running.

pfe1% qstat -s *jobid*
or
pfe1% qstat -f *jobid* | grep -i comment

On Pleiades, output from the following command shows the amount of resources (broken down into Harpertown, Nehalem, and Westmere processors) used and borrowed by each mission directorate, and the resources each mission is waiting for:

pfe1% /u/scicon/tools/bin/qs

The following command provides the order of jobs that PBS schedules to start at the current scheduling cycle. It also provides information regarding processor type(s), mission, and job priority:

```
pfe1% qstat -W o=+model,mission,pri -i
```

The policy described above could result in a large, high-priority job being blocked forever by a steady stream of smaller, low-priority jobs. To prevent jobs from languishing in the queues for an indefinite time, PBS reserves resources for the top N jobs (currently, N is 4), and doesn't allow lower priority jobs start if they would delay the start time of one of the top job ("backfilling"). Additional details are given below.

PBS Sorting Order

- **Mission shares**

  Each NASA mission directorate is allocated a certain percentage of the CPUs in the system. (See Mission Shares Policy on Pleiades .) A job cannot start if that action would cause the mission to exceed its share, unless another mission is using less than its share and has no jobs waiting. In this case, the high-use mission can "borrow" CPUs from the lower-use mission for up to a specified time (currently, max_borrow is 4 hours).

  So , if the job itself needs less than max_borrow hours to run, or if a sufficient number of other jobs from the high-use mission will finish within max_borrow hours to get back under its mission share, then the job can borrow CPUs.

  When jobs are sorted, jobs from missions using less of their share are picked before jobs from missions using more of their share.
- **Job priority**

  Job priority has three components. First is the native priority (the -p parameter to qsub or qalter). Added to that is the queue priority. If the native priority is 0, then a further adjustment is made based on how long the job has been waiting for resources. Waiting jobs get a "boost" of up to 20 priority points, depending on how long they have been waiting and which queue they are in.

  This treatment is modified for queues assigned to the Exploration Systems Mission Directorate (ESMD). For those queues, job priority is set by a separate set of policies controlled by ESMD management.
- **Queue priority**

  Some queues are given higher or lower priorities than the default. (Run "qstat -Q" to get current values.) Note that because the mission share is the most significant sort criterion, job and queue priorities have little effect mission-to-mission.
- **Job size**

Jobs asking for more CPUs are favored over jobs asking for fewer. The reasoning is that, while it is easier for narrow jobs to fill in gaps in the schedule, wide jobs need help collecting enough CPUs to start.

Backfilling

As mentioned above, when PBS cannot start a job immediately, if it is one of the first N such jobs, PBS sets aside resources for the job before examining other jobs. That is, PBS looks at the currently running jobs to see when they will finish (using the wall-time estimates). From those finish times, PBS decides when enough resources (such as CPUs, memory, mission share, and job limits) will become available to run the top job.

PBS then creates a virtual reservation for those resources at that time. Now, when PBS looks at other jobs to see if they can start immediately, it also checks whether starting the job would collide with one of these reservations. Only if there are no collisions will PBS start the lower priority jobs.

This description applies to both Pleiades and Columbia, although the specific queues, priorities, mission percentages, and other elements differ between the two systems.

PBS exit codes

DRAFT

This article is being reviewed for completeness and technical accuracy.

Do we need to have more details for some of the < 0 exit codes? - rh

The PBS exit value of a job may fall in one of four ranges:

- X = 0 (= JOB_EXEC_OK)

  This is a PBS special return value indicating that the job executed successfully
- X < 0

  This is a PBS special return value indicating that the job could not be executed. These negative values are listed below:

  - -1 = JOB_EXEC_FAIL1

    job exec failed, before files, no retry
  - -2 = JOB_EXEC_FAIL2

    job exec failed, after files, no retry
  - -3 = JOB_EXEC_RETRY

    job exec failed, do retry
  - -4 = JOB_EXEC_INITABT

    job aborted on MOM initialization
  - -5 = JOB_EXEC_INITRST

    job aborted on MOM init, checkpoint, no migrate
  - -6 = JOB_EXEC_INITRMG

    job aborted on MOM init, checkpoint, ok migrate
  - -7 = JOB_EXEC_BADRESRT

    job restart failed

    - -8 = JOB_EXEC_GLOBUS_INIT_RETRY

      Init. globus job failed. do retry
    - -9 = JOB_EXEC_GLOBUS_INIT_FAIL

      Init. globus job failed. no retry
- 0 <= X < 128 (or 256 depending on the system)

  This is the exit value of the top process in the job, typically the shell. This may be the exit value of the last command executed in the shell or the .logout script if the user has such a script (csh).
- X >=128 (or 256 depending on the system)

  This means the job was killed with a signal. The signal is given by X modulo 128 (or 256). For example an exit value of 137 means the job's top process was killed with signal 9 (137 % 128 = 9).

Front-End Usage Guidelines

Pleiades Front-End Usage Guidelines

DRAFT

This article is being reviewed for completeness and technical accuracy.

The front-end systems pfe[1-12] and bridge[1,2] provide an environment that allows you to get quick turnaround while performing the following:

- file editing
- compiling
- short debugging and testing session
- batch job submission to the compute systems

Bridge[1,2], with 4 times the memory on pfe[1-12] and better interconnects, can also be used for the following two functions:

1. **Post processing**

   These nodes have 64-bit versions of IDL, Matlab, and Tecplot installed and have 64 GB of memory (4 times the amount of memory on pfe[1-12]). The bridge nodes will run these applications much faster than on pfe[1-12].

2. **File transfer between Pleiades and Columbia or Lou**

   Note that both the Pleiades Lustre filesystems (/nobackupp[10-70]) and the Columbia CXFS filesystems (/nobackup1[1-h], /nobackup2[a-i]) are mounted on the bridge nodes.

   To copy files between the Pleiades Lustre and Columbia CXFS filesystems, log in to bridge[1,2] and use the *cp* command to perform the transfer. The 10 Gigabit Ethernet (GigE) connections on the two bridge nodes are faster than the 1 GigE used on pfe[1-12], therefore, file transfer out of Pleiades is improved when using the bridge nodes.

   File transfers from bridge[1,2] to Lou[1,2] will go over the 10 GigE interface by default. The commands *scp*, *bbftp*, and *bbscp* are available to do file transfers. Since *bbscp* uses almost the same syntax as *scp*, but performs faster than *scp*, we recommend using *bbscp* over *scp* in cases where you do not require the data to be encrypted when sent over the network.

```
The pfe systems ([pfe1-12]) have a 1 GigE connection, which
can be saturated by a single secure copy (scp). You will see
bad performance whenever more than one file transfer is
happening. Use of bridge1 and bridge2 for file transfers is
strongly recommended.
```

File transfers from the compute nodes to Lou must go through pfe[1-12] or bridge[1,2] first, although going through bridge[1,2] is preferred for performance consideration. See Transferring Files from the Pleiades Compute Nodes to Lou for more information.

When sending data to Lou[1-2], please keep your largest individual file size under 1 TB, as large files will keep all of the tape drives busy, preventing other file restores and backups. To prevent the filesystems on Lou[1-2] from filling up, please limit total data transfers to 1 TB and then wait an hour before continuing. This allows the tape drives to write the data to tape.

Additional restrictions apply to using these front-end systems:

1. No MPI jobs are allowed to run on pfe[1-12], bridge[1,2]
2. A job on pfe[1-12] should not use more than 8 GB. When it does, a courtesy email is sent to the owner of the job.
3. A job on bridge[1,2] should not use more than 56 GB. When it does, a courtesy email is sent to the owner of the job.

Columbia Front-End Usage Guidelines

DRAFT

This article is being reviewed for completeness and technical accuracy.

The front-end system, cfe2, provide an environment that allows users to get quick turnaround while performing the following: file editing; file management; short debugging and testing sessions; and batch job submission to the compute systems.

Running long and/or large (in terms of memory and/or number of processors) debugging or production jobs interactively or in the background of cfe2 is considered to be inconsiderate behavior to the rest of the user community. If you need help submitting such jobs to the batch systems, please contact a NAS scientific consultant at (650) 604-4444 or 1-800-331-USER or send e-mail to: support@nas.nasa.gov

Jobs that cause significant impact on the system load of the Columbia front-end machine (cfe2) are candidates for removal in order to bring the front-end systems back to a normal and smooth environment for all users. A *cron* job regularly monitors the system load and determines if job removal is necessary. The criteria for job removal are described below. Owners of any removed jobs will receive a notification e-mail.

1. To be eligible for removal, the number of processors a front-end interactive job uses can be one (1) or more. Exceptions to this are those programs, utilities, etc. common to users and/or NASA missions that are listed in an "exception file". Examples of these would be:

   bash cp csh emacs gzip rsync scp sftp sh ssh tar tcsh

   Users can submit program names to be added to this exception file by mailing requests to: support@nas.nasa.gov
2. For qualifying processes, the CPU time usage of each process in a job has, on the average, exceeded a threshold defined as:

   (20 min x 8 / number of processes for the job)

   That is, a baseline for removal is a job with 8 processors running for more than 20 minutes. The maximum amount of time allowed for each processor in a job is scaled using the formula:

   20 min x 8 cpu / number-of-processes

   Therefore, the following variations are possible:

   - 160 minutes = (20 * 8) / 1 cpu

- 80 minutes = (20 * 8) / 2 cpu
- 40 minutes = (20 * 8) / 4 cpu
- 20 minutes = (20 * 8) / 8 cpu
- 10 minutes = (20 * 8) / 16 cpu
- 5 minutes = (20 * 8) / 32 cpu
- 2.5 minutes = (20 * 8) / 64 cpu

The conditions of removal are subject to change, when necessary.

PBS on Pleiades

Overview

Overview

On Pleiades, PBS (version 10.4) is used to manage batch jobs that run on the compute nodes (3 different processor types, 9,984 nodes and 91,136 cores in total). PBS features that are common to all NAS systems are described in other articles. Read the following articles for Pleiades-specific PBS information:

- queue structure

- resource request examples
- default variables set by PBS
- sample PBS scripts

Queue Structure

Users should be aware of the PBS queue structure. To find the maximum and default NCPUS (number of CPUs), the maximum and default wall time, the priority of the queue, and whether the queue is disabled or stopped, use the command:

```
%qstat -Q
```

This command also provides statistics of jobs in the states of queued (Q), held (H), running (R), or exiting (E).

Note that the queue structure will change from time to time. Below is a snapshot of the output from this command on June 16, 2011.

```
%qstat -Q
Queue    Ncpus/      Time/         State counts
name       max/def    max/def   jm T/_Q/H/W/__R/E/B pr  Info
======== =====/=== ======/===== == ================ === ========
normal      --/  8  08:00/01:00 -- 0/20/4/0/_60/0/0   0
debug     1025/  8  02:00/00:30 -- 0/_3/0/0/__4/0/0  15
low         --/  8  04:00/00:30 -- 0/_0/0/0/__0/0/0 -10
long      8192/  8 120:00/01:00 -- 0/_8/1/0/206/0/0   0
route       --/  8      --/   -- -- 0/_0/0/0/__0/0/0   0
idle        --/ --      --/   -- -- 0/_0/0/0/__0/0/0   0 disabled
alphatst    --/ -- 120:00/01:00 -- 0/_0/0/0/__0/0/0   0
ded_time    --/ --      --/01:00 -- 0/_0/0/0/__0/0/0   0
devel     4800/  1  02:00/   -- -- 0/_1/0/0/__5/0/0   0
wide        --/ -- 120:00/01:00 -- 0/_1/0/0/__0/0/0  45 disabled
testing     --/ --      --/00:30 -- 0/_0/0/0/__0/0/0   0
somd_spl    --/  8 240:00/01:00 -- 0/_0/0/0/__2/0/0  25
armd_spl  4900/  8 120:00/01:00 10 0/_0/0/0/__0/0/0  15
normal_N    --/  8 120:00/01:00 -- 0/_5/0/0/_10/0/0   0
rtf         --/  8      --/01:00 -- 0/_0/0/0/__0/0/0  65
dpr         --/  8      --/00:10 -- 0/_0/0/0/__0/0/0   0
normal_W    --/  8 120:00/01:00 -- 0/60/5/0/_18/0/0   0
S1848368   744/  1  04:00/   -- -- 0/_0/0/0/__0/0/0   0
kepler      --/  8 120:00/01:00 -- 0/12/0/0/_33/0/0  20
diags       --/ -- 120:00/01:00 -- 0/_0/0/0/__0/0/0   0
```

The *devel* queue on Pleiades is served by pbspl3 (a non-default PBS server). The *devel* queue requires pbspl3 to be included in the syntax for qsub, qstat, and qdel. For more information, read the article Pleiades devel Queue.
To view more information about each queue, use:

```
%qstat -fQ queue_name
```

In the output of this command, you will find additional information such as:

acl_groups
   Lists all GIDs that are allowed to run in the queue.
   For the *normal, debug, long, low* and *wide* queues, all GIDs should be included.
   Special queues, such as *esmd_spl, armd_spl, somd_spl, clv_spl*, etc., typically allow

a few GIDs only.

default_chunk.model
: Specifies the default processor model, if you do not specify the processor model yourself.
All queues, except *normal_N* and *normal_W*, default to using nodes with Harpertown model processors.

resources_min.ncpus
: If defined, specifies the minimum NCPUs required for the queue.
The *wide* queue requires using a minimum of 1024 CPUs.

max_user_run
: If defined, specifies the maximum number of jobs each user is allowed to run in the queue.
For the *normal* queue, it is set at 10. For the *debug* queue, it is set at 2.

`The` *`normal_N`* `and` *`normal_W`* `queues will be removed in the near future. To request using the Nehalem-EP or Westmere nodes, use "model=neh" or "model=wes" attribute in your` *`resource_list`*`. To explicitly request Harpertown nodes, use "model=har". You can apply the model attribute to any queue.`

For example:

```
#PBS -q long
#PBS -l select=1:ncpus=12:model=wes
```

Mission Shares Policy on Pleiades

DRAFT

This article is being reviewed for completeness and technical accuracy.

Mission Directorate shares have been implemented on Pleiades since Feb. 10, 2009. Implementing shares guarantees that each Mission Directorate gets its fair share of resources.

The share to which a job is assigned is based on the GID used by the job. Once all the cores within a Mission Directorate's share have been assigned, other jobs assigned to that share must wait, even if cores are available in a different Mission Directorate's share, with the following exception:

When a Mission Directorate is not using all of its cores, other Mission Directorates can borrow those cores, but only for jobs that will finish within 4 hours. When part of the resource is unavailable, the total number of cores decreases, and each Mission Directorate loses a proportionate number of cores.

You can display the share distribution by adding the "-W shares=-" option to the qstat command. For example:

```
%qstat -W shares=-

Group   Share% Use%  Share Exempt   Use  Avail Borrowed Ratio Waiting
------- ------ ---- ------ ------ ----- ------ -------- ----- -------
Overall    100    0 159748      0   960 158788        0  0.01     960
 ARMD       24   18  38109      0 29680   8429        0  0.78   22512
 HEOMD      23   21  36521      0 34312   2209        0  0.94   28416
 SMD        51   50  80981      0 80968     13        0  1.00  113920
 NAS         2    0   3175      0     0   3175        0  0.00   20240
```

Mission shares are calculated by combining the mission's HECC share of the shared assets combined with the mission-specific assets. The mission shares on Oct 3, 2011 are shown in the second column of the above display. The amount of resources used and borrowed by each mission and resources each mission is waiting for are also displayed.

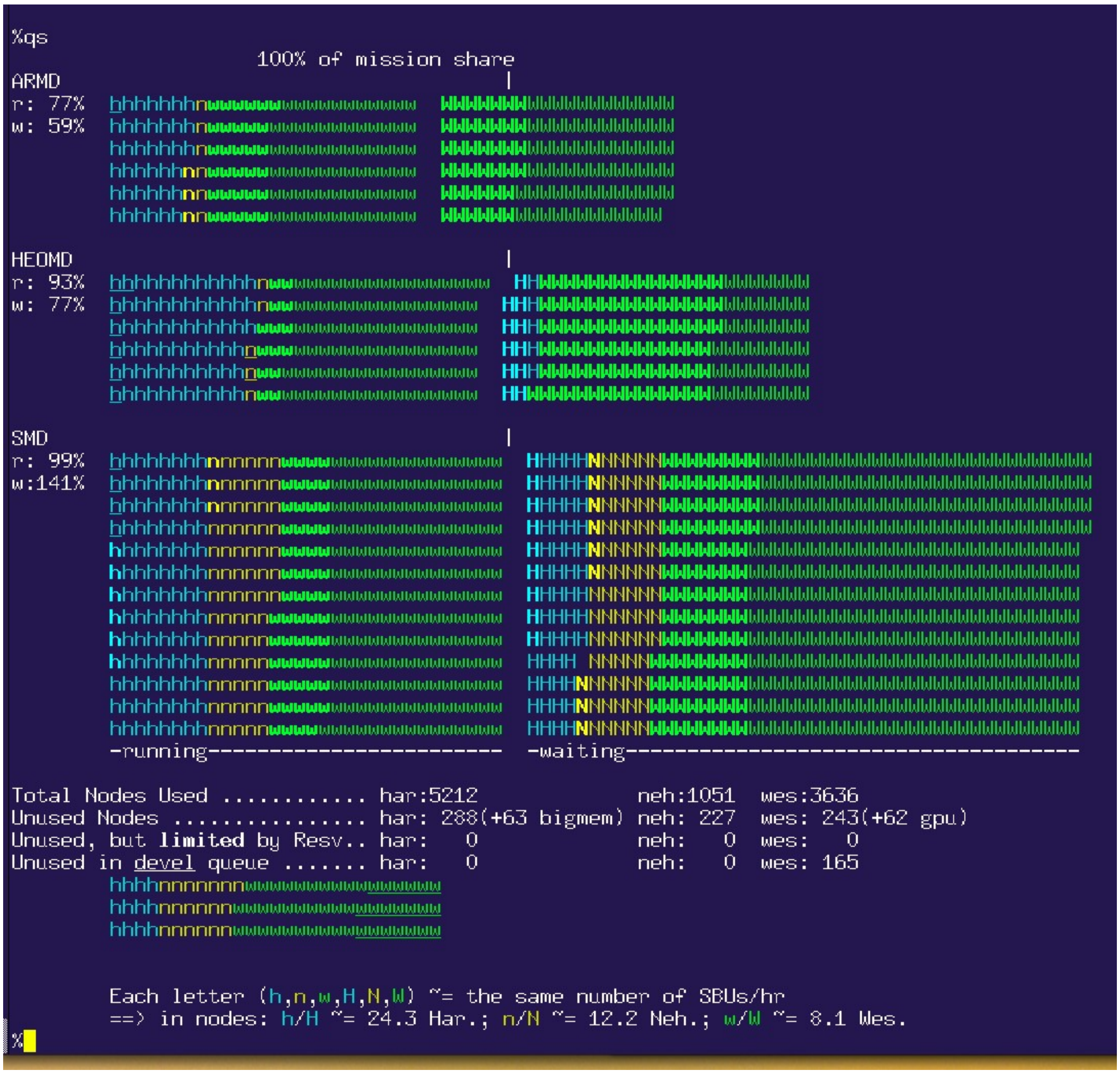
An in-house utility, *qs*, provide similar information with details that break the resources into the Harpertown, Nehalem-EP and Westmere-EP processor types and is available at /u/scicon/tools/bin/qs.

The -h option of qs provides instructions on how to use it:

```
% /u/scicon/tools/bin/qs -h
```

```
usage: qs [-u] [-n N] [-b] [-p] [-d] [-r] [-f M,N] [-q N] [-t] [-v] [-h] [--file f]

          -u       : show used resources only; don't show queued jobs

          -n N     : show time remaining before N nodes are free

          -b       : order segments in bars to help understand borrowing

          -p       : plain output: i.e. no colors or highlights

          -d       : darker colored resource bars (for a light background)

          -r       : use Reverse video for displaying resource bars

          -f M,N   : highlight nodes for jobs that finish in <= M minutes

                     and <= N minutes [default M=60,N=240]

                     (0 turns off highlighting)

          -q N     : highlight nodes for jobs queued in last N minutes [3]

                     (0 turns off highlighting)

          -t       : show time remaining & nodes used for each running job

          --file f : reserved for debugging

          -v       : (verbose) provide explanation of display elements

          -h       : provide this message
```

Here is a sample output file of *qs*:



```
%qs
                    100% of mission share
ARMD                                        |
r: 77%   hhhhhhhnwwwwwwwwwwwwwwwwwwwwwwwwwww  WWWWWWWWWWWWWWWWWWW
w: 59%   hhhhhhhnwwwwwwwwwwwwwwwwwwwwwwwwwww  WWWWWWWWWWWWWWWWWWW
         hhhhhhhnwwwwwwwwwwwwwwwwwwwwwwwwwww  WWWWWWWWWWWWWWWWWWW
         hhhhhhnnwwwwwwwwwwwwwwwwwwwwwwwwwww  WWWWWWWWWWWWWWWWWWW
         hhhhhhnnwwwwwwwwwwwwwwwwwwwwwwwwwww  WWWWWWWWWWWWWWWWWWW
         hhhhhhnnwwwwwwwwwwwwwwwwwwwwwwwwwww  WWWWWWWWWWWWWWWWWW

HEOMD                                       |
r: 93%   hhhhhhhhhhhhnwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHWWWWWWWWWWWWWWWWWWWWWW
w: 77%   hhhhhhhhhhhhnwwwwwwwwwwwwwwwwwwwwwwwwwwwwww   HHHWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhhhhhwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww   HHHWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhhhhnwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww   HHHWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhhhhnwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww   HHHWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhhhhnwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww   HHWWWWWWWWWWWWWWWWWWWWWWW

SMD                                         |
r: 99%   hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
w:141%   hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhnnnnnnwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhnnnnnwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhnnnnnwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhnnnnnwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHHH NNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhnnnnnwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhnnnnnwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
         hhhhhhhhnnnnnwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww  HHHHNNNNNNWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
         -running------------------------  -waiting---------------------------------

Total Nodes Used ............ har:5212             neh:1051  wes:3636
Unused Nodes ................ har: 288(+63 bigmem) neh: 227  wes: 243(+62 gpu)
Unused, but limited by Resv.. har:   0             neh:   0  wes:   0
Unused in devel queue ....... har:   0             neh:   0  wes: 165
         hhhhnnnnnnnwwwwwwwwwwwwwwwwwwwwwwwwww
         hhhhnnnnnnwwwwwwwwwwwwwwwwwwwwwwwwww
         hhhhnnnnnnwwwwwwwwwwwwwwwwwwwwwwwwww


         Each letter (h,n,w,H,N,W) ~= the same number of SBUs/hr
         ==> in nodes: h/H ~= 24.3 Har.; n/N ~= 12.2 Neh.; w/W ~= 8.1 Wes.
%
```

Resources Request Examples

Since Pleiades consists of three different processor types, Harpertown, Nehalem-EP and Westmere-EP, keep the following in mind when requesting PBS resources for your job:

- Starting May 1, 2011, charging on the usage of the three Pleiades processor types is based on a common Standard Billing Unit which is on a per-node basis. The SBU rate for each of the Pleiades processor type is:

  ```
  Processor Type        SBU Rate (per node)
  Westmere-EP           1    (12 cores in a node)
  Nehalem-EP            0.8  (8 cores in a node)
  Harpertown            0.45 (8 cores in a node)
  ```

  The actural amount of memory per node through PBS is slightly less, 7.6 GB/node for Harpertown and 22.5 GB/node for Nehalem-EP and Westmere-EP.

  Use the "model=[har,neh,wes]" attribute to request the processor type(s) for your job. If the processor type is not specified in user's PBS resource list, the job is routed to use the default processor type, Harpertown.

  Example 1:

  Here are some examples of requesting certain processor models for a 128-process job:

  ```
  #PBS -l select=16:ncpus=8:model=har
  # to run all 8 cores on each of 16 Harpertown nodes

  #PBS -l select=32:ncpus=4:model=har
  # to run on only 4 cores on each of 32 Harpertown nodes
  # (note: will be charged for 32 nodes = 256 cores)

  #PBS -l select=16:ncpus=8:model=neh
  # to run all 8 cores on each of 16 Nehalem-EP nodes

  #PBS -l select=11:ncpus=12:model=wes
  # to run all 12 cores on each of 11 Westmere-EP nodes
  # (4 cores in 11th node will go unused)
  ```

  Note that you can specify both the queue type (-q normal, debug, long, wide, low) and the processor type (-l model=har,neh,wes). For example:

  ```
  #PBS -q normal
  #PBS -l select=16:ncpus=8:model=neh
  ```

  If your application can run on any of the three processor types, you may want to submit your job to a processor type that has more unoccupied nodes by other running jobs. This can possibly reduce the wait time of your job. The script *node_stats.sh* provides information about the total, used and free nodes for each processor type. For example:

```
%/u/scicon/tools/bin/node_stats.sh


 Pleiades Nodes Total: 9394
 Pleiades Nodes Used : 8128
 Pleiades Nodes Free : 1266

 Harpertown Total: 5854 Used: 4878 Free: 976
 Nehalem    Total: 1255 Used: 1036 Free: 219
 Westmere   Total: 2285 Used: 2214 Free: 71

Currently queued jobs are requesting: 1463 Harpertown, 1767 Nehalem,
2860 Westmere nodes
```

```
Add "/u/scicon/tools/bin" to your PATH in .cshrc or other
shell start up scripts to avoid having to type in the complete
path for this tool.
```

You can also find for each job what processor models are used by looking at the "Model" field of the output of the command:

```
%qstat -a -W o=+model
```

- The Harpertown nodes in rack 32 have 16 GB memory/node instead of 8 GB/node.

  Example 2:

  This example shows a request of 2 nodes with bigmem in rack 32.

  ```
  #PBS -l select=2:ncpus=8:bigmem=true:model=har
  ```

- For a multi-node PBS job, the NCPUs used in each node can be different. This is useful for jobs that need more memory for some processes, but less for other processes. Resource requests can be done in "chunks" for a job with varying NCPUs per node.

  Example 3:

  This example shows a request of two chunks of resources. In the first chunk, 1 CPU in 1 node, and in the second chunk, 8 CPUs in each of three other nodes are requested:

  ```
  #PBS -l select=1:ncpus=1+3:ncpus=8
  ```

- A PBS job can run across different processor types. This can be useful in two scenarios:

    1. when you can not find enough free nodes within one model for your job
    2. when some of your processes need more memory while others need much less

  This can be accomplished by specifying the resources in "chunks", with one chunk asking for one processor type while another chunk asking for a different processor type.

  Example 4

  Here is an example to request 1 Westmere node (which provides 24 GB/node) and 3 Harpertown nodes (which provides 8 GB/node).

```
#PBS -lplace=scatter:excl:group=model
#PBS -lselect=1:ncpus=12:mpiprocs=12:model=wes+3:ncpus=8:mpiprocs=8:model=har
```

Default Variables Set by PBS

DRAFT

This article is being reviewed for completeness and technical accuracy.

You can use the "env" command--either in a PBS script or on the command line of a PBS interactive session--to find out what environment variables are set within a PBS job. In addition to the PBS_xxxx variables, the following two are useful to know:

- NCPUS defaults to number of CPUs that you requested for the node.
- OMP_NUM_THREADS defaults to 1 unless you explicitly set it to a different number.

  If your PBS job runs an OpenMP or MPI/OpenMP application, this variable sets the number of threads in the parallel region.
- FORT_BUFFERED defaults to 1.

  Setting this variable to 1 enables records to be accumulated in the buffer and flushed to disk later.

Sample PBS Script for Pleiades

DRAFT

This article is being reviewed for completeness and technical accuracy.

```
#PBS -S /bin/csh
#PBS -N cfd
# This example uses the Harpertown nodes
# User job can access ~7.6 GB of memory per Harpertown node.
# A memory intensive job that needs more than ~0.9 GB
# per process should use less than 8 cores per node
# to allow more memory per MPI process. This example
# asks for 64 nodes and 4 MPI processes per node.
# This request implies 64x4 = 256 MPI processes for the job.
#PBS -l select=64:ncpus=8:mpiprocs=4:model=har
#PBS -l walltime=4:00:00
#PBS -j oe
#PBS -W group_list=a0801
#PBS -m e

# Currently, there is no default compiler and MPI library set.
# You should load in the version you want.
# Currently, MVAPICH or SGI's MPT are available in 64-bit only,
# you should use a 64-bit version of the compiler.

module load comp-intel/11.1.046
module load mpi-sgi/mpt.2.04.10789

# By default, PBS executes your job from your home directory.
# However, you can use the environment variable
# PBS_O_WORKDIR to change to the directory where
# you submitted your job.

cd $PBS_O_WORKDIR

# use of dplace to pin processes to processors may improve performance
# Here you request to pin processes to processors 2, 3, 6, 7 of each node.
# This helps for using the Harpertown nodes, but not for Nehalem-EP or
# Westmere-EP nodes

# The resource request of select=64 and mpiprocs=4 implies
# that you want to have 256 MPI processes in total.
# If this is correct, you can omit the -np 256 for mpiexec
# that you might have used before.

mpiexec dplace -s1 -c2,3,6,7 ./grinder < run_input > output

# It is a good practice to write stderr and stdout to a file (ex: output)
# Otherwise, they will be written to the PBS stderr and stdout in /PBS/spool,
# which has limited amount  of space. When /PBS/spool is filled up, any job
# that tries to write to /PBS/spool will die.
# -end of script-
```

Pleiades devel Queue

NAS provides a special *devel* queue that provides faster turnaround when doing development work.

Currently, 512 Westmere nodes are reserved for the Pleiades *devel* queue, 24x7. The maximum wall-time allowed is 2:00:00 and the maximum NCPUS is 4800. Each user is allowed to have only one job running in the *devel* queue at any one time.

To improve PBS job scheduling response time, the *devel* queue and its resources (for example, nodes) have been moved to a second PBS server (pbspl3). With this move, users must specify the server name for jobs managed by pbspl3 with qsub, qstat, and qdel if the command is launched from a Pleiades front-end node (pfe[1-12] or bridge[1-4]). For example:

```
pfe1% qsub -q devel@pbspl3 job_script
1234.pbspl3.nas.nasa.gov

pfe1% qstat devel@pbspl3

pfe1% qstat -a @pbspl3

pfe1% qstat -u zsmith @pbspl3

pfe1% qstat 1234.pbspl3

pfe1% qdel 1234.pbspl3
```

Alternatively, if you set the environment variable **PBS_DEFAULT** to pbspl3, you can skip pbspl3 in your qsub, qstat, qdel commands. For example (in csh):

```
pfe1% setenv PBS_DEFAULT pbspl3

pfe1% qsub -q devel job_script
1234.pbspl3.nas.nasa.gov

pfe1% qstat devel

pfe1% qstat -a

pfe1% qstat -u zsmith

pfe1% qstat 1234

pfe1% qdel 1234
```

Use the csh command *unsetenv PBS_DEFAULT* to return to using the default PBS server, pbspl1.

Note that the changes described here do not apply to jobs submitted to the other queues (normal, long, debug, and all special queues) served by the default server, pbspl1.

To see all jobs you have submitted to pbspl1 or pbspl3 (using username zsmith in the example below), type the following:

```
pfe1% qstat @pbspl1 @pbspl3 -W combine -u zsmith
```

PBS on Columbia

Overview

DRAFT

This article is being reviewed for completeness and technical accuracy.

On Columbia, PBS (version 9.2) is used to manage batch jobs that run on the four compute systems (Columbia21-24). PBS features that are common to all NAS systems are described in other articles. Read the following articles for Columbia-specific PBS information:

- queue structure

- resource request examples
- default variables set by PBS
- sample PBS scripts

Resources Request Examples

DRAFT

This article is being reviewed for completeness and technical accuracy.

All of the Columbia compute engines, Columbia21-24, are single system image Altix 4700 systems:

```
Columbia21 (508 CPUs total, 1.8 GB memory/CPU through PBS)
Columbia22 (2044 CPUs total, 1.8 GB memory/CPU through PBS)
Columbia23 (1020 CPUs total, 1.8 GB memory/CPU through PBS)
Columbia23 (1020 CPUs total, 1.8 GB memory/CPU through PBS)
```

Here are a few examples of requesting resources on Columbia:

Example 1:

If your job needs fewer than 508 CPUs and you do not care which Columbia system to run your job on, simply use *ncpus* to specify the number of CPUs that you want for your job. For example:

```
#PBS -l ncpus=256
```

Example 2:

If you specify both the *ncpus* and *mem* for your job, PBS will make sure that your job is allocated enough resources to satisfy both *ncpus* and *mem*. For example, if you request 4 CPUs and 14 GB of memory, your job will be allocated 8 CPUs and 14.4 GB because the amount of memory associated with 4 CPUs is not enough to satisfy your memory request.

```
#PBS -l ncpus=4,mem=14GB
```

Example 3:

If you want your job to run on a specific Columbia machine, for example, Columbia22 with 256 CPUs, use

```
#PBS -l select=host=columbia22:ncpus=256
```

Note that the ncpus request must appear with the select=host request and must not be present as a separate request either on the qsub command line or in the PBS script.
Example 4:

If you ever need to run a job across two Columbia systems, for example, 508 CPUs on one Columbia and another 508 CPUs on another, use

```
#PBS -l select=2:ncpus=508,place=scatter
```

Default Variables Set by PBS

DRAFT

This article is being reviewed for completeness and technical accuracy.

You can use the "env" command--either in a PBS script or from the command line of an interactive PBS session--to find out what environment variables are set within a PBS job. In addition to the PBS_xxxx variables, the following ones are useful to know.

- NCPUS defaults to number of CPUs that you requested.
- OMP_NUM_THREADS defaults to 1 unless you explicitly set it to a different number.

  If your PBS job runs an OpenMP or MPI/OpenMP application, this variable sets the number of threads in the parallel region.
- OMP_DYNAMIC defaults to *false*.

  If your PBS job runs an OpenMP application, this disables dynamic adjustment of the number of threads available for execution of parallel regions.
- MPI_DSM_DISTRIBUTE defaults to *true*.

  If your PBS job runs an MPI application, this ensures that each MPI process gets a unique CPU and physical memory on the node with which that CPU ist is associated.
- FORT_BUFFERED defaults to 1.

  Setting this variable to 1 enables records to be accumulated in the buffer and flushed to disk later.

Sample PBS Script for Columbia

DRAFT

This article is being reviewed for completeness and technical accuracy.

```
#PBS -S /bin/csh
#PBS -N cfd
#PBS -l ncpus=4
#PBS -l mem=7776MB
#PBS -l walltime=4:00:00
#PBS -j oe
#PBS -W group_list=g12345
#PBS -m e

# By default, PBS executes your job from your home directory.
# However, you can use the environment variable
# PBS_O_WORKDIR to change to the directory where
# you submitted your job.

cd $PBS_O_WORKDIR

# For MPI jobs, there is an SGI MPT module loaded by default, unless you
# modify your shell start up script to unload it or switch to a different
# version. You can use either mpiexec or mpirun to start your job.

mpiexec -np 4 ./a.out < input > output

# It is a good practice to write stderr and stdout to a file (ex: output)
# Otherwise, they will be written to the PBS stderr and stdout in /PBS/spool
# which has limited amount  of space. When /PBS/spool is filled up, any job
# that tries to write to /PBS/spool will die.

# -end of script-
```

Troubleshooting PBS Jobs

Common Reasons for Being Unable to Submit Jobs

DRAFT

This article is being reviewed for completeness and technical accuracy.

There are several common reasons why you might not be able to successfully submit a job to PBS:

- Resource request exceeds resource limits

  qsub: Job exceeds queue resource limits

  Reduce your resource request to below the limit or use a different queue.
- AUID or GID not authorized to use a specific queue

  If you get the following message after submitting a PBS job:

  qsub: Unauthorized Request

  it is possible that you tried submitting to a queue which is accessible only to certain groups or users. You can check the "qstat -fQ" output and see if there is an acl_groups or a acl_users list. If your group or username is not in the lists, you will have to submit to a different queue."

- AUID not authorized to use a specific GID

  If you get the following message after submitting a PBS job:

  qsub: Bad GID for job execution

  it is possible that your AUID has not been added to use allocations under a specific GID. Please consult with the principal investigator of that GID and ask him/her to submit a request to support@nas.nasa.gov to add your AUID under that GID.
- Queue is disabled

  If you get the following message after submitting a PBS job

  qsub: Queue is not enabled

  you should submit to a different queue which is enabled.

- Not enough allocation left

  An automated script is used to check if a GID is over its allocation everyday. If it does, that GID is removed from PBS access control list and users of that GID will not be able to submit jobs.

  Users can check the amount of allocations remaining using the acct_ytd command. In addition, if you see in your PBS output file some message regarding your GID allocation usage is near its limit or is already over, ask your PI to request for more allocation.

Common Reasons Why Jobs Won't Start

Once you've successfully submitted your job, there may be several common reasons why it might not run:

- **The job is waiting for resources**

  Your job may be waiting for resources, due to one of the following:
  - All resources are tied up with running jobs, and no other jobs can be started.
  - PBS may have enough resources to run your job, however, there is another higher priority job that needs more resources than what is available, and PBS is draining the system (including not running any new jobs) in order to accommodate the high-priority job.
  - Some users submit too many jobs at once (e.g., more than 100), and the PBS scheduler becomes swamped with sorting jobs and is not able to start jobs effectively.
  - In the case when you request your job to run on a specific node or group of nodes, your job is likely to wait in the queue longer than if you do not request specific nodes.
- **Your mission share has run out**

  Your mission shares have been used up. The available resources that you saw belong to other missions, which can be borrowed. However, your job may have requested a wall-time that is too long (more than 4 hours for Pleiades), which prevents your job from borrowing the resources.

  See also, Mission Shares Policy on Pleiades.
- **The system is going into dedicated time**

  When dedicated time (DED) is scheduled for hardware and/or software work, the PBS scheduler will not start a job if the projected end time runs past the beginning of the DED time. If you are able to reduce the requested wall-time so that your job will finish running prior to DED time, then your job can then be considered for running. To change the wall-time request for your job, follow the example below :

  ```
  %qalter -l walltime=hh:mm:ss jobid
  ```
- **Scheduling is turned off**

  Sometimes job scheduling is turned off by control room staff or a system administrator. This is usually done when there are system or PBS issues that need to be resolved before jobs can be scheduled to run. When this happens, you should see the following message near the beginning of the "qstat -au your_userid" output.

  ```
  +++Scheduling turned off.
  ```
- **Your job has been placed in "H" mode**

A job can be placed on hold either by the job owner or by someone who has root privilege, such as a system administrator. If your job has been placed on hold by a system administrator, you should get an email explaining the reason for the hold.

- **Your home filesystem or default /nobackup filesystem is down**

  When a PBS job starts, the PBS prologue checks to determine whether your home filesystem and default /nobackup are available before executing the commands in your script. If your default /nobackup filesystem is down, PBS can not run your job and it will put the job back in the queue. If your PBS job does not need any file in that filesystem, you can tell PBS that your job will not use the default /nobackup so that your job can start running. For example, if your default is /nobackupp10 (for Pleiades), you can add the following in your PBS script:

  ```
  #PBS -l /nobackupp10=0
  ```

Using pdsh_gdb for Debugging Pleiades PBS Jobs

DRAFT

This article is being reviewed for completeness and technical accuracy.

A script called *pdsh_gdb*, created by NAS staff Steve Heistand, is available on Pleiades under */u/scicon/tools/bin* for debugging PBS jobs **while the job is running**.

Launching this script from a Pleiades front-end node allows one to connect to each compute node of a PBS job and create a stack trace of each process. The aggregated stack trace from each process will be written to a user specified directory (by default, it is written to ~/tmp).

Here is an example of how to use this script:

```
pfe1% mkdir tmp
pfe1% /u/scicon/tools/bin/pdsh_gdb -j jobid -d tmp -s -u nas_username
```

More usage information can be found by launching pdsh_gdb without any option:

```
pfe1% /u/scicon/tools/bin/pdsh_gdb
```

Effective Use of PBS

Streamlining File Transfers from Pleiades Compute Nodes to Lou

Some users prefer to streamline the storage of files (created during a job run) to Lou, within a PBS job. Since Pleiades compute nodes do not have network access to the outside world, all file transfers to Lou within a PBS job must go through the front-ends (pfe[1-12], bridge[1,2]) first.

Here is an example of what you can add to your PBS script to accomplish this:

1. Ssh to a front-end node (for example, bridge2) and create a directory on Lou where the files are to be copied.

   ```
   ssh -q bridge2 "ssh -q lou mkdir -p $SAVDIR"
   ```
   Here, $SAVDIR is assumed to have been defined earlier in the PBS script. Note the use of *-q* for quiet-mode, and double quotes so that shell variables are expanded prior to the *ssh* command being issued.

2. Use scp via bridge[1,2] to transfer the files.

   ```
   ssh -q bridge2 "scp -q $RUNDIR/* lou:$SAVDIR"
   ```
   Here, $RUNDIR is assumed to have been defined earlier in the PBS script.

Avoiding Job Failure from Overfilling /PBS/spool

Before a PBS job is completed, its error and output files are kept in the /PBS/spool directory of the first node of your PBS job. The space under /PBS/spool is limited, however, and when it fills up, any job that tries to write to /PBS/spool may die. To prevent this, you should *not* write large amount of contents in the PBS output/error files.

If your executable normally produces a lot of output to the screen, you should redirect its output in your PBS script. For example:

```
#PBS ...
mpiexec a.out > output
```

To see the contents of your PBS output/error files before your job completes, follow the two steps below:

1. Find out the first node of your PBS job using "-W o=+rank0" for qstat:

   ```
   %qstat -u your_username -W o=+rank0
   JobID         User   Queue  Jobname   TSK Nds    wallt S    wallt  Eff Rank0
   ------------- ------ ------ -------- ---- --- -------- - -------- ---- ---------
   868819.pbspl1 zsmith long   ABC       512  64 5d+00:00 R 3d+08:39 100% r162i0n14
   ```

   This shows that the first node is r162i0n14.

2. Log in to the first node and *cd* to /PBS/spool to find your PBS stderr/out file(s). You can view the content of these files using *vi* or *view*.

```
%ssh r162i0n14
%cd /PBS/spool
%ls -lrt
-rw------- 1 zsmith a0800 49224236 Aug  2 19:33 868819.pbspl1.nas.nasa.gov.OU
-rw------- 1 zsmith a0800  1234236 Aug  2 19:33 868819.pbspl1.nas.nasa.gov.ER
```

Running Multiple Serial Jobs to Reduce Walltime

DRAFT

This article is being reviewed for completeness and technical accuracy.

On Pleiades, running multiple serial jobs within a single batch job can be accomplished with following example PBS scripts. The maximum number of processes you can run on a single node will be limited to the core-count-per-node or the maximum number that will fit in a given node's memory, whichever is smaller.

```
processor type  cores/node      available memory/node
 Harpertown         8                     7.6 GB
 Nehalem-EP         8                    22.5 GB
 Westmere-EP       12                    22.5 GB
```

The examples below allow you to spawn serial jobs accross nodes using the mpiexec command. Note that a special version of mpiexec from the mpi-mvapich2/1.4.1/intel module is needed in order for this to work. This mpiexec keeps track of $PBS_NODEFILE and places each serial job onto the CPUs listed in $PBS_NODEFILE properly. The use of the arguments "-comm none" for this version of mpiexec is essential for serial codes or scripts. In addition, to launch multiple copies of the serial job at once, the use of the mpiexec-supplied $MPIEXEC_RANK environment variable is needed to distinguish different input/output files for each serial job. This is demonstrated with the use of a wrapper script "wrapper.csh" in which the input/output identifier (i.e., ${rank}) is calculated from the sum of $MPIEXEC_RANK and an argument provided as input by the user.

Example 1:

This first example runs 64 copies of a serial job, assuming that 4 copies will fit in the available memory on one node and 16 nodes are used.

serial1.pbs:

```
#PBS -S /bin/csh
#PBS -j oe
#PBS -l select=16:ncpus=4
#PBS -l walltime=4:00:00

module load mpi-mvapich2/1.4.1/intel

cd $PBS_O_WORKDIR

mpiexec -comm none -np 64 wrapper.csh 0
```

wrapper.csh:

```
#!/bin/csh -f
@ rank = $1 + $MPIEXEC_RANK
./a.out < input_${rank}.dat > output_${rank}.out
```

This example assumes that input files are named input_0.dat, input_1.dat, ... and that they are all located in the directory where the PBS script is submitted from (i.e., $PBS_O_WORKDIR). If the input files are in different directories, then wrapper.csh can be modified appropriately to cd into different directories as long as the directory names are differentiated by a single number that can be obtained from $MPIEXEC_RANK (=0, 1, 2, 3, ...). In addition, be sure that wrapper.csh is executable by you and you have the current directory included in your path.

Example 2:

A second example provides the flexibility where the total number of serial jobs may not be the same as the total number of CPUs requested in a PBS job. Thus, the serial jobs are divided into a few batches and the batches are processed sequentially. Again, the wrapper script is used where multiple versions of the program "a.out" in a batch are run in parallel.

serial2.pbs:

```
#PBS -S /bin/csh
#PBS -j oe
#PBS -l select=10:ncpus=3
#PBS -l walltime=4:00:00

module load mpi-mvapich2/1.4.1/intel

cd $PBS_O_WORKDIR

# This will start up 30 serial jobs 3 per node at a time.
# There are 64 jobs to be run total, only 30 at a time.

# The number to run in total defaults here to 64 or the value
# of PROCESS_COUNT that is passed in via the qsub line like:
# qsub -v PROCESS_COUNT=48 serial2.pbs
#

# the total number to run at once is automatically determined
# at runtime by the number of cpus available.
# qsub -v PROCESS_COUNT=48 -l select=4:ncpus=3 serial2.pbs
# would make this 12 per pass not 30. no changes to script needed.

if ( $?PROCESS_COUNT ) then
 set total_runs=$PROCESS_COUNT
else
 set total_runs=64
endif

set batch_count=`wc -l < $PBS_NODEFILE`

set count=0
```

```
while ($count < $total_runs)
  @ rank_base = $count
  @ count += $batch_count
  @ remain = $total_runs - $count
  if ($remain < 0) then
    @ run_count = $total_runs % $batch_count
  else
    @ run_count = $batch_count
  endif
  mpiexec -comm none -np $run_count wrapper.csh $rank_base
end
```

Checking the Time Remaining in a PBS Job from a Fortran Code

DRAFT

This article is being reviewed for completeness and technical accuracy.

During job execution, sometimes it is useful to find out the amount of time remaining for your PBS job. This allows you to decide if you want to gracefully dump restart files and exit before PBS kills the job.

If you have an MPI code, you can call MPI_WTIME and see if the elapsed walltime has exceeded some threshold to decide if the code should go into the shutdown phase.

For example,

```
include "mpif.h"

real (kind=8) :: begin_time, end_time

begin_time=MPI_WTIME()
do work
end_time = MPI_WTIME()

if (end_time - begin_time > XXXXX) then
   go to shutdown
endif
```

In addition, the following library has been made available on Pleiades for the same purpose:

/u/scicon/tools/lib/pbs_time_left.a

To use this library in your Fortran code, you need to:

1. Modify your Fortran code to define an external subroutine and an integer*8 variable

   ```
   external pbs_time_left
   integer*8 seconds_left
   ```
2. Call the subroutine in the relevant code segment where you want the check to be performed

   ```
   call pbs_time_left(seconds_left)
   print*,"Seconds remaining in PBS job:",seconds_left
   ```

```
       The return value from pbs_time_left is only  accurate to within a minute or
```

3. Compile your modified code and link with the above library using, for example

```
LDFLAGS=/u/scicon/tools/lib/pbs_time_left.a
```